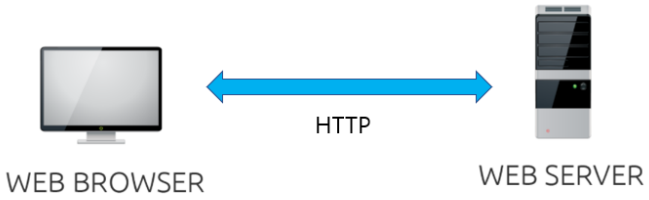
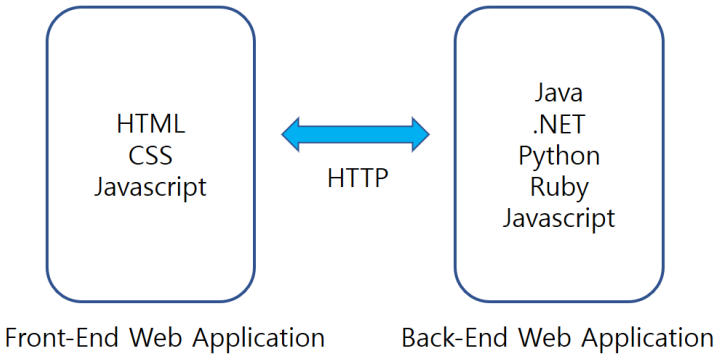
25장. 자바스크립트 개발을 위한 코틀린

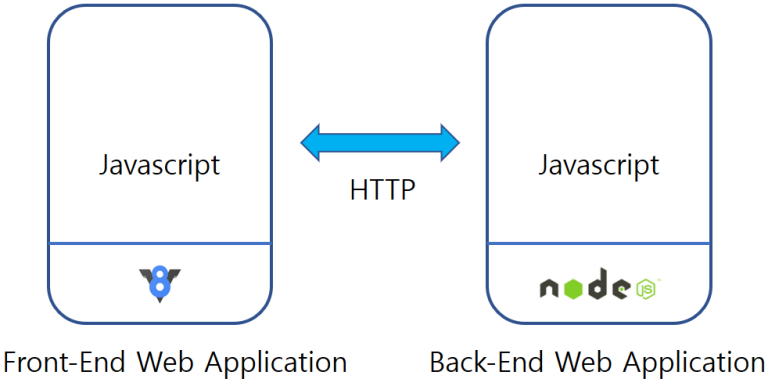
25.1. 자바스크립트 이해

25.1.1. 자바스크립트 개발 환경에 대한 이해





25.1. 자바스크립트 이해



25.1. 자바스크립트 이해

25.1.2. 자바스크립트 개발 언어

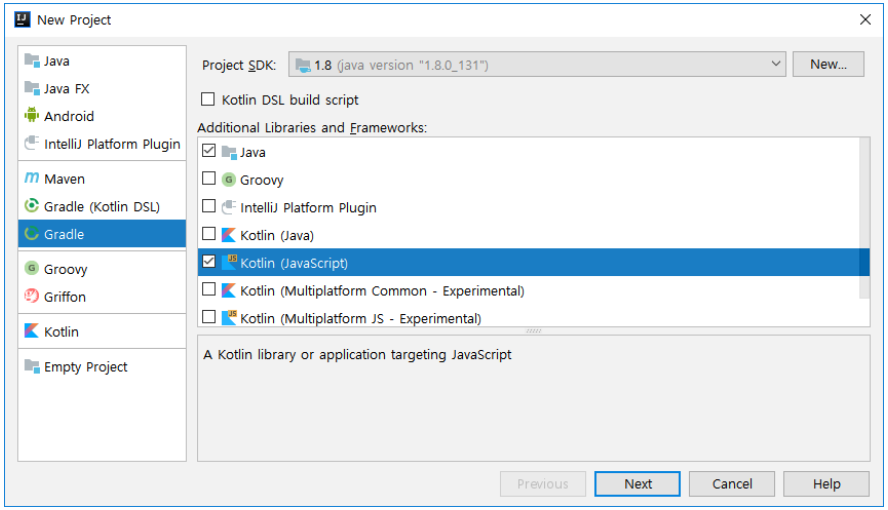
- 자바스크립트는 ECMA 단체에 의해 표준이 정의되는 언어

- 변수를 선언할 수는 있지만 데이터 타입이 지원되지 않는다.
- 함수는 함수의 매개변수를 맞추지 않아도 함수명만 맞으면 호출이 된다.
- Prototype 이라는 독특한 방법으로 클래스를 정의한다.
- 모듈화 프로그램이 불가능하다.

- ECMA에서 새로운 표준을 정의하여 ECMAScript6 발표
- TypeScript
- Swift

25.2. 자바스크립트를 위한 코틀린 개발환경

25.2.1. IntelliJ에서 Gradle을 이용한 자바스크립트 개발환경

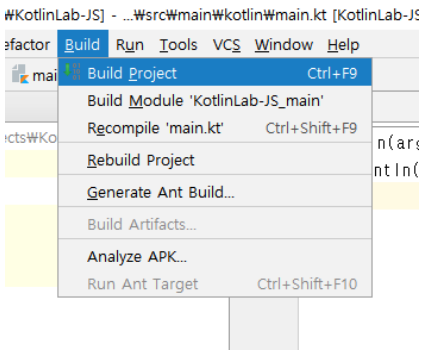


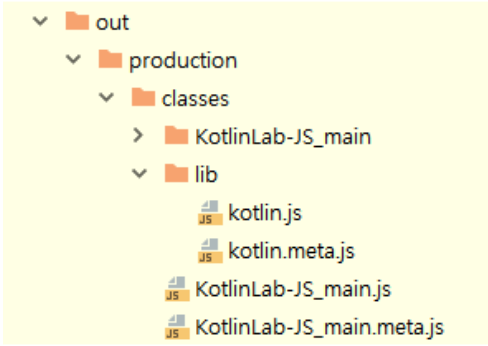

```
//.............
apply plugin: 'java'
apply plugin: 'kotlin2js'

//.............
dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib-js:$kotlin_version"
    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```

25.2. 자바스크립트를 위한 코틀린 개발환경

```
fun main(args: Array<String>) {
    println("Hello World")
}
```






```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<script type="text/javascript" src="../out/production/classes/lib/kotlin.js"></script>
<script type="text/javascript" src="../out/production/classes/KotlinLab-JS_main.js"></script>

</body>
</html>
```

25.2. 자바스크립트를 위한 코틀린 개발환경

25.2.2. Command Line 을 이용한 자바스크립트 개발

- kotlinc-js -output result.js main.kt

```
C:₩Temp>kotlinc-js -output result.js main.kt

C:₩Temp>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: D20D-A6C1

 C:₩Temp 디렉터리

2017-12-10  오후 12:25    <DIR>          .
2017-12-10  오후 12:25    <DIR>          ..
2017-12-10  오전 11:36                62 main.kt
2017-12-10  오후 12:25               490 result.js
               2개 파일                 552 바이트
               2개 디렉터리  303,127,470,080 바이트 남음
```

25.3. 자바스크립트를 위한 코틀린 기법

25.3.1. dynamic type

- 자바스크립트의 경우는 변수에 데이터 타입을 명시하지 않는다.
- 데이터 타입을 지정 할수 있다는 점은 자바스크립트를 코클린으로 작성했을때의 이점

```
var data1=10
data1="kkang"

var resultDom= document.getElementById("result_3_1_js")
resultDom.innerHTML="data1 : "+data1;
```

```
import kotlin.browser.document

data class User(val no: Int, val name: String)


fun main(args: Array<String>) {
    val obj = User(10, "kkang")

    val resultDom= document.getElementById("result_3_1_dynamic_1")
    resultDom?.innerHTML="no : ${obj.no}, name : ${obj.name}"

}
```

25.3. 자바스크립트를 위한 코틀린 기법

- dynamic 타입으로 선언하면 자바스크립트 변수와 동일하게 다양한 타입의 데이터가 대입
- 자바스크립트를 목적으로 하는 곳에서만 사용할수 있는 타입
- dynamic 타입으로 선언된 변수는 Null 체크에서도 자유로와 진다.
- 컴파일러에 의해 각 타입의 맴버(변수, 함수)를 체크 불가

```
fun dynFun(arg: dynamic): dynamic{
    return 10
}

var dynData: dynamic = 10
dynData = "hello"

dynFun(10)
dynFun(true)
```

```
val myData2: dynamic = null
val myData3: dynamic? = null
val myData4: dynamic? = 10
```

```
myData2.a1=10
myData2.aaa(10, "hello")
```

25.3. 자바스크립트를 위한 코틀린 기법

25.3.2. js() 함수 이용

- 자바스크립트의 구문을 그대로 코틀린에서 사용

```
var data2
if(data2==undefined){
    alert('undefined')
}else {
    alert('defined')
}
```

```
fun ch25_3_2(){
    var data2: dynamic
    if(data2==undefined){//error
        alert('undefined')//error
    }else {
        alert('defined')//error
    }
}
```

25.3. 자바스크립트를 위한 코틀린 기법

```
js("if(data2==undefined){ alert('undefined') }else { alert('defined') }")
```

```
js("""
if(data2==undefined){
    alert('undefined')
}else {
    alert('defined')
}
    """)
```

25.3. 자바스크립트를 위한 코틀린 기법

25.3.3. external로 자바스크립트 API 이용

- 코틀린에서 자바스크립트 API을 그래도 이용

```
alert("hello kkang")//error
```

```
external fun alert(message: Any?): Unit

fun ch25_3_3(){
    alert("hello kkang")
}
```

- `external val document: Document`

25.3. 자바스크립트를 위한 코틀린 기법

```
var jsData=10
function jsFun(arg){
    var sum=0
    for(i=1; i<=arg; i++){
        sum += i
    }
    console.log('sum:'+sum)
}
var JsClass = (function() {
    var msg
    function JsClass(msg) {
        this.msg = msg;
    }
    JsClass.prototype.sayHello = function() {
        console.log ("hello, " + this.msg)
    }
    return JsClass;
})();
```

```
external val jsData: Int
external fun jsFun(arg:Int): Unit
external class JsClass(msg: String) {
    fun sayHello()
}
fun ch25_3_3(){
    jsFun(jsData)
    val obj=JsClass("kim")
    obj.sayHello()
}
```

25.3. 자바스크립트를 위한 코틀린 기법

클래스 static 멤버를 external로 선언

- 자바스크립트 클래스에 선언된 멤버는 객체 멤버와 클래스 멤버로 구분
- 객체 멤버는 클래스의 객체로 접근하는 함수와 변수를 지칭하며 prototype 예약어로 클래스에 등록
- 클래스 멤버는 객체생성 없이 클래스명으로 접근

- 코틀린에서 자바스크립트의 클래스를 external 로 선언해서 이용할 때 클래스 멤버 함수는 companion object로 선언해야 한다.

```
var JsClass = (function() {
    var msg
    function JsClass(msg) {
        this.msg = msg;
    }
    JsClass.prototype.sayHello = function() {
        console.log("hello, " + this.msg)
    }
    JsClass.sayHello2=function(arg){
        console.log('hello2' + arg)
    }
    return JsClass;
})();
```

25.3. 자바스크립트를 위한 코틀린 기법

```
external class JsClass(msg: String) {
    fun sayHello()
    companion object {
        fun sayHello2(arg: String)
    }
}

fun ch25_3_3(){
    val obj=JsClass("kim")
    obj.sayHello()

    JsClass.sayHello2("lee")
}
```

25.3. 자바스크립트를 위한 코틀린 기법

Optional Parameter를 가지는 함수를 external로 선언

```
function jsFun2(arg1, arg2=10, arg3="hello"){
    console.log(arg1+':'+arg2+':'+arg3);
}
```

```
external fun jsFun2(arg1: String, arg2: Int = 10, arg3: String = "hello"): Unit//error
```

- 코틀린에서 이 매개변수가 기본값이 선언된 매개변수라고 명시적으로 선언

```
external fun jsFun2(arg1: String, arg2: Int = definedExternally, arg3: String = definedExternally): Unit
```

25.3. 자바스크립트를 위한 코틀린 기법

코틀린에서 자바스크립트 클래스를 상속으로 재정의하기

```
var JsSuper = (function() {
    function JsSuper() { }
    JsSuper.prototype.superFun = function() {
        console.log("i am js superFun")
    }
    return JsSuper;
})();
```

```
open external class JsSuper {
    open fun superFun()
}

class Sub: JsSuper() {
    fun subFun(){
        console.log("i am kotlin subFun")
    }

    override fun superFun() {
        super.superFun()
        console.log("i am kotlin superFun")
    }
}
```

25.3. 자바스크립트를 위한 코틀린 기법

external interface 이용

- 자바스크립트에서는 인터페이스라는 개념이 없다.
- 코틀린의 인터페이스 개념을 그래도 적용해서 특정함수의 매개변수를 인터페이스 타입으로 선언 가능

```
function jsFun3(argObj) {
    argObj.argFun1()
    argObj.argFun2()
}
```

```
external interface KotlinInterfaceType{
    fun argFun1()
    fun argFun2()
}
external fun jsFun3(argFun: KotlinInterfaceType)

fun ch25_3_3(){
    jsFun3(object: KotlinInterfaceType{
        override fun argFun1() {
            console.log("argFun1 call....")
        }

        override fun argFun2() {
            console.log("argFun2 call....")
        }
    })
}
```

25.4. 자바스크립트에서 코틀린 소스 이용하기

25.4.1. 모듈명으로 코틀린 소스 이용

```
val myVal="hello myVal"

fun myFun() {
    println("myFun call....")
}

class MyClass {
    fun myClassFun(){
        println("myClassFun call....")
    }
}
```

```
console.log(myVal)//error
```

25.4. 자바스크립트에서 코틀린 소스 이용하기

```
var Ch25_main = function (_, Kotlin) {
  //……….
  var myVal;
  function myFun() {
    println('myFun call....');
  }
  function MyClass() {
  }
  MyClass.prototype.myClassFun = function () {
    println('myClassFun call....');
  };

  //……..  Kotlin.defineModule('Ch25_main', _);
  return _;
}(typeof Ch25_main === 'undefined' ? {} : Ch25_main, kotlin);
```

```
console.log(Ch25_main.myVal)

Ch25_main.myFun()

var obj=new Ch25_main.MyClass()
obj.myClassFun()
```

25.4. 자바스크립트에서 코틀린 소스 이용하기

25.4.2. @JsName 어노테이션 이용

package 구문을 명시해 이용

- package가 선언된 경우 모듈명만 사용해서는 접근 불가

```
package twenty_five_four_two

val someVal="someVal.."
```

```
console.log(Ch25_main.twenty_five_four_two.someVal)
```

25.4. 자바스크립트에서 코틀린 소스 이용하기

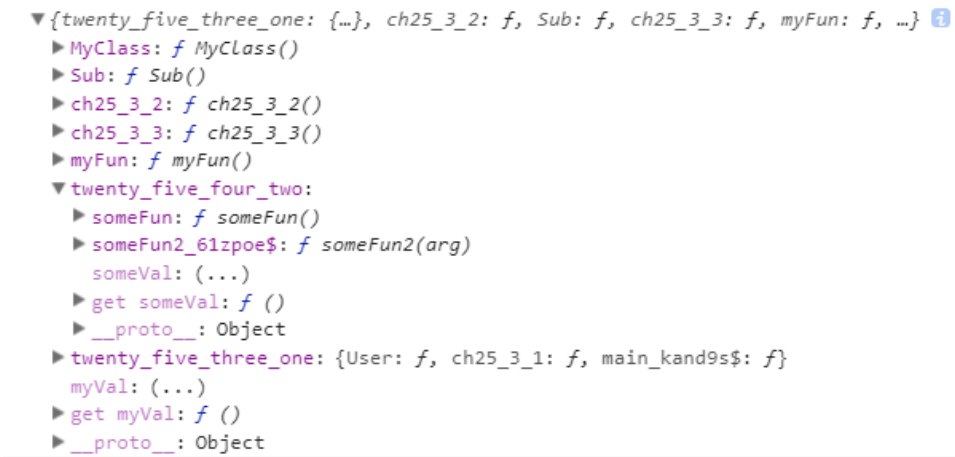
@JsName 으로 함수 이용

- 매개변수를 가지는 함수이용시 주의

```
package twenty_five_four_two

fun someFun(){
    println("someFun call....")
}
fun someFun2(arg: String){
    println("someFun2 call... $arg")
}
```

```
Ch25_main.twenty_five_four_two.someFun()
Ch25_main.twenty_five_four_two.someFun2("kkang") //runtime error
```



25.4. 자바스크립트에서 코틀린 소스 이용하기

- @JsName 어노테이션을 이용해 이름 지정

```
@JsName("helloFun")
fun someFun2(arg: String){
    println("someFun2 call... $arg")
}
```

```
Ch25_main.twenty_five_four_two.helloFun("kkang")
```